

---

# LES NAMESPACES

---

# C++

**Introduction:** Le namespace sert à regrouper une portion de code sous un nom défini. Cela permet d'ajouter de la lisibilité au code et de réduire les problèmes de collision entre fonctions, objets et variables du même nom. Deux fonctions peuvent par exemple porter le même nom à condition de se trouver dans deux namespaces différents.

```
namespace espace {  
    class entier {  
        private : int ent ;  
        public : int GetEnt () const { return ent ; }  
    };  
}
```

Syntaxe de la déclaration d'un namespace

Dans ce code la classe « entier » est incluse au namespace « espace »  
Pour l'accéder, il faudra précéder son nom de celui du namespace à l'aide de l'opérateur de portée (on obtient la visibilité sur le namespace).

```
int main () {  
    espace :: entier objet ;  
    return 0 ;  
}
```

Syntaxe d'accès a une variable du namespace

Afin d'ignorer ce concept de visibilité et de pouvoir utiliser simplement « entier objet », on peut utiliser la directive **using**.

On peut l'utiliser pour le namespace entier ...

```
using namespace espace ;
```

Tout le code qui suit cette déclaration aura une visibilité sur le namespace « espace »

... ou pour un élément uniquement.

```
using espace :: entier ;
```

Dans le code qui suivra cette déclaration, toute utilisation de « entier » sera liée au namespace « espace »

**Remarque :** en ANSI C++, l'entête standard de iostream est <iostream> (et non pas <iostream.h>. Dans <iostream> cout, cin, endl, etc font partie du namespace std et vous obligent donc à taper std ::cout. Afin de vous en affranchir vous pouvez précéder vos programmes de la directive « using namespace std ; » ou « using std :: cout ; », « using std :: cin ; », etc.

Cependant ces déclarations annulent l'intérêt du namespace tout en ramenant le problème de conflit. Il est donc préférable de coder « std::cout » que d'utiliser using.

```

namespace espace {
  class entier {
    private : int ent ;
    public : int GetEnt () const { return ent ; }
  };
}
namespace espace1 {
  class entier {
    private : int ent ;
    public : int GetEnt () const { return ent ; }
  };
}

using namespace espace ; // visibilité sur le namespace « espace »
using namespace espace1 ; // visibilité sur le namespace « espace1 »
int main () {
  entier objet ; // de quel namespace ???
  return 0 ;
}

```

Exemple d'ambiguïté

Les namespaces sont également imbriquables.

```

namespace espace {
  class entier {
    private : int ent ;
    public : int GetEnt () const { return ent ; }
  };
  namespace espace1 {
    class entier {
      private : int ent ;
      public : int GetEnt () const { return ent ; }
    };
  }
}

```

Le namespace « espace1 » est imbriqué dans « espace »

On accédera alors a entier dans espace1 par une imbrication d'opérateurs de portée

```

int main () {
  espace :: espace1 :: entier objet ;
  return 0 ;
}

```

On accède a entier dans « espace1 »

On peut a certains moments trouver que l'imbrication de namespaces ou que le nom lui-même du namespace est trop long et fatiguant à taper. On utilise alors un alias.

```

namespace alias = NomLongBeaucoupTropLong ;
namespace alias1 = espace :: espace1 :: espace2 :: espace3 ;

```

L'alias pourra être dorénavant utilisé à la place de l'expression d'origine